"Express Mail" mailing label number:

EV401038604US

MULTIPLIER STRUCTURE SUPPORTING DIFFERENT PRECISION MULTIPLICATION OPERATIONS

Paul J. Jagodik
Jeffrey S. Brooks
Christopher Olson

BACKGROUND

Field of the Invention

[1001] This application relates arithmetic structures in computer systems and more particularly to structures utilized in multiplication operations.

Description of the Related Art

[1002] Computer architectures may include many different types of multiply instructions with various input and output data precision. For example, single instruction multiple data (SIMD) multimedia instructions are typically implemented using partitioned input and output data. Also, in the same implementation, architectures are demanding large 64 bits multiplies.

[1003] Conventional floating point units (FPUs) support many data flows customized to a specific multiply instruction to create the best possible performance. By increasing the number of types of multiply data flows implemented independently in a micro-architecture causes the area, power, and costs to grow.

[1004] Many multimedia instruction set architectures define several signed and unsigned integer and single precision floating point instructions, which execute as a SIMD (single instruction, multiple data) operation, producing various precision results in parallel.

[1005] It would be desirable to implement the various multiplies required by a particular architecture in a single multiplier structure to provide a better area, power, and cost tradeoff for a performance level.

SUMMARY

[1006] Accordingly, an embodiment of the invention provides for execution of SIMD and non-SIMD multiplies in one multiplier to save area, power, and cost with minimal impact on instruction latency and performance. The multiplies may be both integer and floating point operations.

[1007] In one embodiment, the invention provides a multiplication unit that includes a partitionable multiplier structure partitionable into two or more sub-tree structures. The multiplication unit responds to an indication of a first instruction type to configure the multiplier structure into the two or more sub-trees structure to perform multiple multiplication operations in parallel. The multiplication unit responds to an indication of a second instruction type to configure the sub-tree structure into a single data flow to perform a single multiplication operation.

[1008] In another embodiment, a method is provided for performing a multiplication operation using a partitionable data flow structure having a plurality of sub-trees. The method includes formatting one or more first operands, according to an instruction decode, by formatting at least a least significant bit in each data partition, each data partition corresponding to a sub-tree, that is adjacent to another data partition corresponding to another sub-tree. A second operand is supplied to each sub-tree of the partitionable multiplier structure; and the formatted one or more first operands are used to generate control signals for encoding the second operands to provide partial products supplied in each sub-tree.

[1009] In another embodiment, a method is provided for performing multiplication operations that includes configuring a partitionable tree structure to be a single tree structure in response to a first multiplication instruction decode and as a partitioned tree structure in response to a second multiplication instruction decode; and performing at least two multiplication operations in parallel in the partitioned tree structure in response to the second multiplication instruction decode and performing a single multiplication operation in the single tree structure in response to the first multiplication instruction decode.

BRIEF DESCRIPTION OF THE DRAWINGS

[1010] The present invention may be better understood, and its numerous objects, features, and advantages made apparent to those skilled in the art by referencing the accompanying drawings in which the use of the same reference symbols in different drawings indicates similar or identical items.

[1011] Fig. 1 illustrates various ways to partition a multiplier structure in accordance with an embodiment of the invention to support multiple multiplication operations.

[1012] Fig. 2 illustrates an operand data format in accordance with an embodiment of the invention.

[1013] Fig. 3 illustrates Booth encoded selects according to an embodiment of the invention.

[1014] Fig. 4 illustrates a partitioned multiplier providing for four partitioned 16 bit multiplies according to an embodiment of the invention.

[1015] Fig. 5 illustrates a partitioned multiplier providing for two partitioned 32 bit multiplies according to an embodiment of the invention.

[1016] Fig. 6 illustrates a partitionable multiplier structure configured for a 64 bit multiply according to an embodiment of the invention.

[1017] Fig. 7 illustrates the use of the multiplier structure for four 16 bit multiplies.

[1018] Fig. 8 illustrates the use of the multiplier structure for two 32 bit multiplies.

[1019] Fig. 9 illustrates the use of the multiplier structure for a 64 bit multiply.

DESCRIPTION OF THE PREFERRED EMBODIMENT(S)

[1020] Referring to Fig. 1, illustrated are the various ways that a partitionable multiplier structure can be configured according to one embodiment of the invention

that supports both SIMD and non-SIMD instructions. The multiplier structure 100 can be partitioned into multiple partitions for SIMD multiplies and into a single data flow for non-SIMD (e.g., a scalar type) multiply instructions. In the exemplary embodiment illustrated in Fig. 1, the partitionable structure 100 can be partitioned into a maximum of four equally sized partitions, each partition being similar to tree D, partition 103. In other embodiments, the maximum number of partitions may be greater than or less than four. When configured into four partitions for execution of SIMD multiply instructions, the data flow can accommodate four multiplications executed concurrently. Additionally, the four partitions can operate on independent data. The partitionable tree structure can also be configured into two partitions, each similar to tree AB, partition 105, which can be used to multiply 32 bit data. When the four partitions are combined (tree ABCD) the single data flow 107 can multiply 64 bit data for non-SIMD instructions. In one embodiment, a different SIMD or scalar type instruction can be executed every cycle by the partitionable structure.

[1021] In addition to partitioning into the various trees, the multiplier structure 100 includes a format stage in order to support the partitioning. That is, the input operands are specially formatted to enable the sharing of hardware in the partitionable multiplier structure 100. Fig. 2 illustrates how an input operand, referred to herein as the RS1 operand for convenience, is formatted to create a Booth encoding operand. As is well known in the art, Booth encoding (also referred to as recoding) is utilized to more efficiently multiply data by reducing the number of partial products generated and therefore reducing the number of additions that need to be performed in the multiplier.

[1022] In a conventional multiply operation, a partial product is generated for each bit in the multiplier, which are then summed. With binary arithmetic, the multiplier is either a one or a zero, so each partial product is either the multiplicand (1 X multiplicand) or 0 (0 X multiplicand). The maximum number of partial products that need to be added corresponds to the number of non-zero bits in the multiplier. Booth encoding reduces the number of partial products utilizing multiple bits to determine each partial product. Booth encoding typically uses two current bits as well as the MSB of the previous two bits to determine the partial product. Fig. 2 illustrates the formatting of the multiplier (or Booth operand) used to generate the Booth

encoding as it is formatted for the different types of multiplies. For convenience, the multiplier will be referred to as the RS1 operand.

[1023] Referring to Fig. 2 four multiplexers 201, 203, 205, and 207 are illustrated that are utilized to format the RS1 operand according to the type of instruction that is being decoded. Thus for 64 bit non-SIMD multiply instructions the operand will span the whole RS1 register as indicated by 209 showing that the partitionable structure is a single data flow (tree ABCD). Fig. 3 illustrates the partitioned Booth encoder. Booth encoders 301 and 303 are the first and last encoders utilized by tree D. Booth encoders 305 and 307 are the first and last encoders utilized by tree C. Booth encoders 309 and 311 are the first and least encoders utilized by tree B. Booth encoders 313 and 315 are the first and last encoders utilized by tree D. The partial product controlled by each Booth encoder is indicated in Fig. 3. For example, Booth encoder 301 generates partial product 0 (PP0) and Booth encoder 315 generates partial product 31 (PP31). For the case of a 64 by 64 bit multiply, the 32 partial products (PP0 to PP31) are summed to generate the result.

[1024] For SIMD instructions, the multiple data will be partitioned and formatted into the booth encoder. To avoid data from corrupting a neighboring booth encoder data, a duplicated bit will have to be generated. The duplicated bits will be on the edges of each partitioned sub-tree structure. These edge bits are 47, 31, and 15 as indicated in Fig. 2. Depending on the instruction type the data value of these duplicated bits could be identical, zero, or a sign extension bit. For SIMD instructions having 32 bit multiplies, the operands spanning trees AB 211 and CD 213 are provided to the Booth encoder partitioned for two operands with bit 31 provided to Booth encoder 309 being zeroed out. For SIMD instructions having 16 bit multiplies, the operands spanning trees A 215, B 217, C 219, and D 221, are provided to the Booth encoder partitioned for four operands with bit 15 (provided to Booth encoder 305), bit 31 provided to Booth encoder 309, and bit 47 provided to Booth encoder 313 being zeroed out to ensure that a bit from one partition operand does not corrupt an adjacent operand. Remember that for the non-partitioned utilization of the Booth encoders, encoders 305, 309 and 313 receive as the LSB a bit associated with an adjacent sub-tree that represents the MSB of the previous two bits. When the partitionable structure is partitioned, that bit needs to be zero.

[1025] In other contexts, for example when the partition is for four sixteen bit multiplies, an instruction may specify a signed multiplication with one or both of the operands being less than 16 bits. In such circumstances, the bits supplied to, e.g., Booth encoder 303 may require sign extension. Thus, the formatting performed by multiplexers 201-207 and any additional formatting logic for sign extension that is not explicitly shown, is dependent upon the instruction decode. The instruction decode specifies whether a non-SIMD scalar type multiplication (e.g. 64 bit operands) or a SIMD multiplication (e.g. 32 or 16 bit operands). The instruction decode may specify, e.g., a double precision multiplication operation, a single precision operation, an integer multiply, and may specify whether the operation is signed or unsigned. Thus, e.g., the instruction decode can specify whether sign extension is required because of a signed multiply having an operand used for the Booth encoding that is less than 16 bits.

[1026] Referring to Fig. 4, an embodiment is illustrated in which the partitionable data flow is partitioned into four sub-trees 401, 403, 405, and 407, which provide for four sixteen bit multiplies in parallel. Thus, a SIMD multiply is performed by partitioning the data flow tree into four sections and formatting the input and output data appropriately. Note that the non-Booth operand, designated as RS2 herein, is duplicated in register 409, 411, 413 and 415 for each sub-tree. Duplicating the register in each sub-tree allows a partitioned operand data type to enter the multiplier tree. The registers each support 64 bit multiplications and thus are wider than needed for 16 or 32 bit multiplies. Input multiplexers 441, 443, 445, and 447 supply respective RS2 registers with appropriately formatted RS2 data. The selects for the multiplexers will be generated based on the instruction type or decode. For example, for signed sixteen bit multiplies, the data supplied to the RS2 registers will be sign extended. For unsigned multiplies, the RS2 registers may be zeroed out above sixteen bits. In one embodiment the precision and sign of the RS1 and/or RS2 operands may vary for data specified by the SIMD instruction. Note that RS1 data may also be formatted appropriately to reflect signed and/or unsigned operations.

[1027] A 16 bit operand multiplied by a 16 bit operand will produce a 32 bit result. The four multiplication results are supplied in sum and carry representation on nodes 421, 423, 425 and 427. Those results are supplied to and selected by

multiplexers 121 and 123 when the instruction specifies 16 bit SIMD operations. Thus, the instruction decode determines the multiplexer selects for multiplexers 121 and 123. Note that Fig. 1 shows all the data that is selectable being supplied to multiplexers 121 and 123, while Figs. 4-6 show only the data being utilized in the particular embodiment being supplied to the multiplexer, thus implicitly showing the multiplexer selection control. Note that the adder 125 can be used to place the result in non-redundant representation (as opposed to sum and carry). The adder needs to ensure that additions performed for four 16 bit multiply results are performed independently. That is, carries from one result should not affect an adjacent result. That may be accomplished, for example, by suppressing carries or spacing the data for each multiply result with a bit between the results.

[1028] Fig. 5 illustrates how the tree structure illustrated in Fig. 1 is partitioned to multiply two partitioned 32 bit data values. By adding an additional level of carry-save-adder (CSA), shows as CSAs 501 and 503, and appropriately formatting the input operands and appropriately selecting the output data, the Wallace tree structure can multiply two partitioned 32 bit values. The data flow is shown as Wallace tree AB 505, and tree CD 507. The two multiplication results are supplied by CSA's 501 and 503. Those results are selected by multiplexers 121 and 123 with the selects being determined according to the type of instruction being executed.

[1029] Fig. 6 illustrates how the tree structure illustrated in Fig. 1 is utilized to multiply 64 bit data value. By adding an additional level of carry-save-adder (CSA) 601, the Wallace tree can multiply two 64 bit values. The data flow is shown as Wallace Tree ABCD 603.

[1030] The output format multiplexer has to take data from different stages of the Wallace tree depending on the output precision of the result. For example, for 16 bit multiples, the output format bypasses any additional carry-save-adders which would corrupt the final result data.

[1031] The utilization of the partitionable multiplier structure is further illustrated in Figs. 7 – 9. Fig. 7 illustrates how the Wallace tree or carry save adder structures are utilized for four 16 bit multiplies. Referring to Figs. 4 and 7, the partial products PP00 through PP07 are supplied by Booth multiplexers (represented symbolically by

multiplexers 431 and 433) to CSAs 435 and 437. Fig. 8 illustrates how the structures are utilized for two 32 bit multiply operations and Fig. 9 illustrates how the Wallace tree structure is utilized for a 64 bit multiply operation.

[1032] The embodiments described above are presented as examples and are subject to other variations in structure and implementation within the capabilities of one reasonably skilled in the art. The details provided above should be interpreted as illustrative and not as limiting. Variations and modifications of the embodiments disclosed herein, may be made based on the description set forth herein, without departing from the scope of the invention as set forth in the following claims.